Texture mapping progressive meshes

Texture Mapping Progressive Meshes

Pedro V. Sander
Harvard University
http://cs.harvard.edu/~pvs

John Snyder
Microsoft Research
johnsny@microsoft.com

Steven J. Gortler
Harvard University
http://cs.harvard.edu/~sjg

Hugues Hoppe
Microsoft Research
http://research.microsoft.com/~hoppe

Abstract

Given an arbitrary mesh, we present a method to construct a progressive mesh (PM) such that all meshes in the PM sequence share a common texture parametrization. Our method considers two important goals simultaneously. It minimizes texture stretch (small texture distances mapped onto large surface distances) to balance sampling rates over all locations and directions on the surface. It also minimizes texture deviation ("slippage" error based on parametric correspondence) to obtain accurate textured mesh approximations. The method begins by partitioning the mesh into charts using planarity and compactness heuristics. It creates a stretch-minimizing parametrization within each chart, and resizes the charts based on the resulting stretch. Next, it simplifies the mesh while respecting the chart boundaries. The parametrization is re-optimized to reduce both stretch and deviation over the whole PM sequence. Finally, the charts are packed into a texture atlas. We demonstrate using such atlases to sample color and normal maps over several models.

Additional Keywords: mesh simplification, surface flattening, surface parametrization, texture stretch.

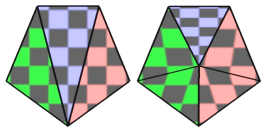
1. Introduction

The progressive mesh (PM) representation encodes an arbitrary mesh as a simple base mesh M^0 and a sequence of n refinement operations called vertex splits [10]. It defines an array $\{M^0 \ldots M^n\}$ of level-of-detail (LOD) approximations, and supports geomorphs and progressive transmission [1]. Unlike multiresolution frameworks based on subdivision, the meshes in a PM have irregular connectivities that can accurately model sharp features (e.g. creases and corners) at all scales.

One challenge in the PM framework is dealing with texture maps. Hardware rasterization features (including bump maps, normal maps, and multitexturing) let fine detail be captured in texture images parametrized over the mesh. Sources for textures include sampling detailed scanned meshes, evaluating solid textures, ray tracing, and 3D painting. In this paper, we address the problem of parametrizing texture images over all meshes in a PM sequence.

A single unfolding of an arbitrary mesh onto a texture image may create regions of high distortion, so generally a mesh must be partitioned into a set of *charts*. Each chart is parametrized by a region of a texture domain, and these parametrizations collectively form an *atlas* (see Figure 4c). For instance, several schemes [2][22][25][27] simplify a mesh and then construct a texture image chart over each simplified face by sampling attributes (e.g. normals) from the original mesh.

For a PM, one might then consider re-using chart images defined on faces of M^0 for all meshes $M^1 \ldots M^n$. However, the problem is that a PM is generally not *chart-compliant*, in that its vertex splits can change the chart topology when applied indiscriminately near chart boundaries, thereby forcing parametric discontinuities. For example, the vertex split shown on the right changes the adjacency of the three colored charts, resulting in the discontinuous texture. Fortunately, it is possible to construct a single atlas parametrization for the entire PM sequence. Chart-compliance can be obtained by first defining the charts on the original mesh, and then *constraining* a simplification sequence to comply with those chart boundaries [3].



Therefore, our problem is the following: given an arbitrary mesh, parametrize it onto a texture atlas, and create a PM sequence compliant with the atlas charts. In doing so, we have two goals:

- Minimize **texture stretch**: The parametrization determines sampling density over the surface, but is constructed before knowing what texture map(s) will be applied. Therefore, we seek a balanced parametrization rather than one that samples finely in some surface regions or directions while undersampling others. A conservative, local measure of how finely the parametrization samples the texture signal is the larger singular value of its Jacobian, which measures how much a sampling direction in the texture domain is stretched on the mesh surface in the worst case. By minimizing the largest texture stretch across all domain points, we create a balanced parametrization where no domain direction is too stretched and thus undersamples its corresponding mapped 3D direction. (See Figure 1 and Figure 6.)

- Minimize **texture deviation**: Traditional mesh simplification measures geometric error by approximating closest-point (Hausdorff) distance. For textured surfaces, it is more appropriate to use the stricter *texture deviation* error, which measures geometric error according to parametric correspondence [3]. For a PM, texture deviation can be graphed as a function of mesh complexity (Figure 3). Our goal is to lower this graph curve.

Recall that the motivation for partitioning the surface into charts is to reduce texture stretch. However, the presence of chart boundaries hinders simplification quality since chart-compliance requires that these boundaries appear as edges in all meshes including M^0. In the extreme, if each face of M^n is made its own chart, stretch is zero, but no simplification can occur. Hence, there exists a trade-off between texture stretch and deviation.

Minimizing stretch and deviation is a difficult nonlinear problem over both discrete and continuous variables. The discrete variables are the mesh partition and the edge collapse sequence. The continuous variables are the texture coordinates of the vertices. Our approach is to set the discrete variables early, using heuristics, and then proceed to optimize the continuous variables. Specifically, our method has the following steps:

(1) partition original mesh into charts (considering geometry)
(2) form initial chart parametrizations (minimizing stretch)
(3) resize chart polygons (based on stretch)
(4) simplify mesh (minimizing texture deviation, creating PM)
(5) optimize parametrization (stretch & deviation over *all* PM)
(6) pack chart polygons (forming texture atlas)
(7) sample texture images (using atlas parametrization)

The contributions of our work are:

- an algorithm for partitioning a mesh into charts, which considers simplification quality and does not alter the mesh.
- a texture stretch metric that uniformly penalizes undersampling everywhere over the surface.
- an algorithm for minimizing this stretch metric in the L^2 and L^∞ norms, which can be used for both static meshes and PMs.
- a scheme for optimizing the parametrization to minimize both texture stretch and texture deviation at *all* PM levels, with appropriate weighting of each mesh in $M^0 \ldots M^n$.
- the first automatic solution for creating a PM representation with a consistent surface parametrization for all LODs.

2. Previous work

Mesh partitioning into charts. Several authors have proposed methods for parametrizing meshes by partitioning into charts. Krishnamurthy and Levoy [17] describe an interactive system in which the user manually lays out chart boundaries by tracing curves. Maillot et al. [21] partition mesh faces according to a bucketing of face normals. Eck et al. [4] use a Voronoi-based partition. These last two algorithms make little effort to adapt charts to surface geometry, so the chart boundaries can hinder simplification, leading to poor LOD approximations.

MAPS [18] and Normal Meshes [8] map edges of the simplified base domain back to the original mesh. While the resulting charts adapt to surface geometry, their boundaries cut across faces of original mesh, requiring addition of new vertices and faces. For the applications in [8][18], these additional vertices are only temporary, because the mesh geometry is subsequently resampled. However, our application is to generate a PM from a user-specified mesh, whose connectivity is often carefully optimized, so the imposition of new vertices is a drawback.

Chart parametrization. Several schemes have been proposed to flatten surface regions to establish a parametrization. The schemes typically obtain the parametrization by minimizing an objective functional. The main distinction between the functionals is how they measure the distance of the parametrization from an isometry (a mapping preserving lengths and angles).

Maillot et al. [21] base their metric on edge springs of nonzero rest length, where rest length corresponds to edge length on the surface. To ensure that the parametrization is 1-to-1, (i.e., to avoid parametric "buckling", also called "face flipping"), they add an area-preservation term to the metric. When the texture domain boundary is fixed as in our application, it is unclear how edge rest-lengths should be scaled. More importantly, the weighting between the edge springs and the area-preservation term must be adjusted to produce an embedding.

Eck et al. [4] propose the harmonic map, which weights edge springs non-uniformly. The weights can sometimes be negative, in which case an embedding is not guaranteed. Floater [5] proposes a similar scheme with a different edge-spring weighting that guarantees embedding for convex boundaries. For either method, the parametrization can be found by solving a linear system.

Lévy and Mallet [19] combine orthogonality and isoparametric terms in their metric. To solve the resulting nonlinear optimization, they iteratively fix one texture component (*s* or *t*) and solve for the other using a linear optimization. As in [20], a term is added which must be sufficiently weighted to guarantee an embedding.

Hormann and Greiner [11] propose the MIPS parametrization, which roughly attempts to preserve the ratio of singular values over the parametrization. However, the metric disregards absolute stretch scale over the surface, with the result that small domain areas can map to large regions on the surface.

To allow all meshes in the PM to share a common texture map, it is necessary that we create domains with straight boundaries between chart corners, unlike [11][19][21].

Our main contribution is to directly optimize the two relevant goals for texture mapping PMs: minimal texture stretch and minimal texture deviation. Our novel stretch metric attempts to balance sampling rates everywhere on the surface, unlike previous techniques.

Appearance-preserving simplification. Cohen et al. [3] introduce texture deviation as the appropriate measure of geometric accuracy when simplifying textured meshes. The texture deviation between a simplified mesh M^i and the original mesh M^n at a point $p^i \in M^i$ is defined as $\| p^i - p^n \|$ where p^n is the point on M^n with the same parametric location in the texture domain. Cohen et al. track texture deviation conservatively by storing a bounding error box at each mesh vertex. They demonstrate results on parametric surfaces already organized into charts.

We begin with an unparametrized mesh, and seek to form an atlas parametrization that specifically minimizes texture deviation and stretch over all meshes in a PM.

3. Texture stretch metric

To optimize a parametrization's ability to balance frequency content everywhere over the surface in every direction, we define a new "texture stretch" metric on triangle meshes.

Given a triangle T with 2D texture coordinates p_1, p_2, p_3, $p_i = (s_i, t_i)$, and corresponding 3D coordinates q_1, q_2, q_3, the unique affine mapping $S(p) = S(s,t) = q$ is

$$S(p) = \big(\langle p, p_2, p_3\rangle q_1 + \langle p, p_3, p_1\rangle q_2 + \langle p, p_1, p_2\rangle q_3\big) \big/ \langle p_1, p_2, p_3\rangle$$

where $\langle a,b,c\rangle$ denotes area of triangle *abc*. Since the mapping is affine, its partial derivatives are constant over (*s,t*) and given by

$$S_s = \partial S/\partial s = \big(q_1(t_2 - t_3) + q_2(t_3 - t_1) + q_3(t_1 - t_2)\big)\big/(2A)$$

$$S_t = \partial S/\partial t = \big(q_1(s_3 - s_2) + q_2(s_1 - s_3) + q_3(s_2 - s_1)\big)\big/(2A)$$

$$A = \langle p_1, p_2, p_3\rangle = \big((s_2 - s_1)(t_3 - t_1) - (s_3 - s_1)(t_2 - t_1)\big)\big/2$$

The larger and smaller singular values of the Jacobian $[S_s, S_t]$ are given respectively by

$$\Gamma = \sqrt{1/2\left((a+c) + \sqrt{(a-c)^2 + 4b^2}\right)} \quad \text{max singular value}$$

$$\gamma = \sqrt{1/2\left((a+c) - \sqrt{(a-c)^2 + 4b^2}\right)} \quad \text{min singular value}$$

where $a = S_s \cdot S_s$, $b = S_s \cdot S_t$, and $c = S_t \cdot S_t$. The singular values Γ and γ represent the largest and smallest length obtained when mapping unit-length vectors from the texture domain to the surface, i.e. the largest and smallest local "stretch". We define two stretch norms over triangle T:

$$L^2(T) = \sqrt{\left(\Gamma^2 + \gamma^2\right)/2} = \sqrt{(a+c)/2}, \qquad L^\infty(T) = \Gamma\ .$$

The norm $L^2(T)$ corresponds to the root-mean-square stretch over all directions in the domain, and the worst-case norm $L^\infty(T)$ is the greatest stretch, i.e. the maximum singular value. Note that both $L^2(T)$ and $L^\infty(T)$ increase to infinity as the parametrization of T

becomes degenerate, since its parametric area A drops to zero. If the triangle T flips parametrically (i.e. if A becomes negative), we define both $L^2(T)$ and $L^\infty(T)$ to remain infinite.

We define two analogous norms over the surface of the entire mesh $M = \{T_i\}$:

$$L^2(M) = \sqrt{\sum_{T_i \in M} \left(L^2(T_i)\right)^2 A'(T_i) \Big/ \sum_{T_i \in M} A'(T_i)}$$

$$L^\infty(M) = \max_{T_i \in M} L^\infty(T_i)$$

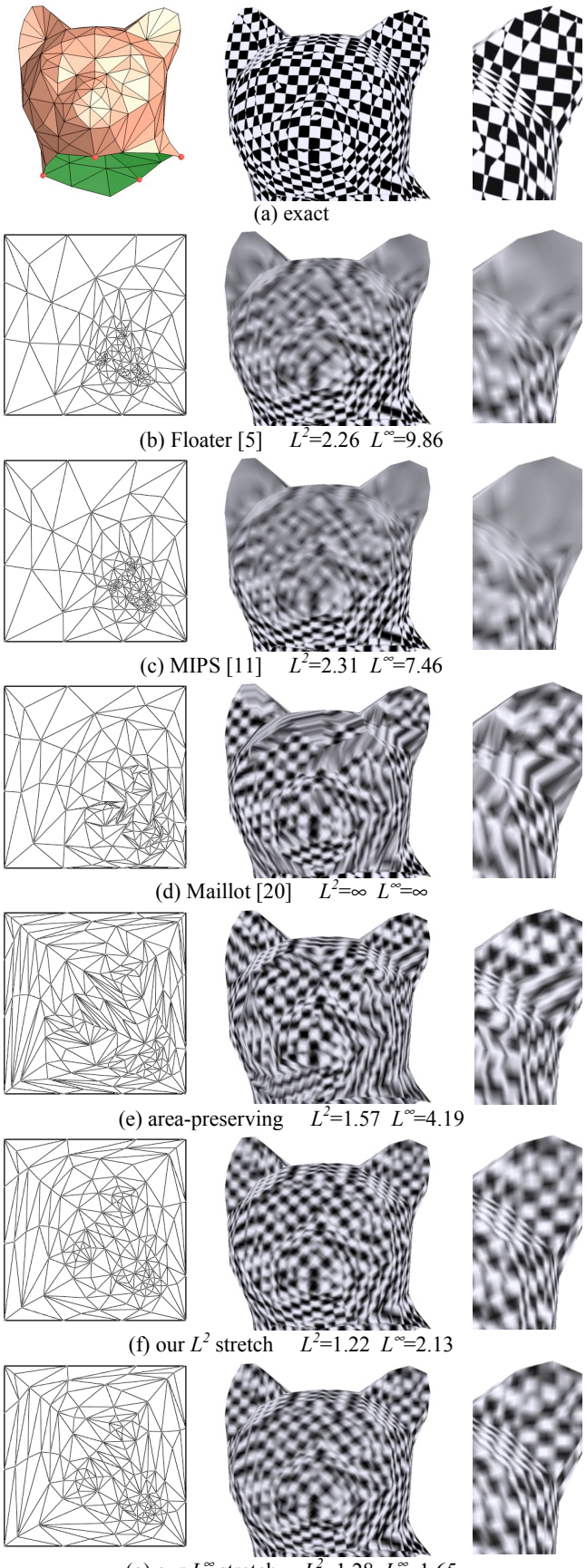
where $A'(T_i)$ is the surface area of triangle T_i in 3D. The L^2 norm measures the overall ability of the parametrization to support high-frequency textures, while L^∞ measures its worst-case ability.

We normalize stretch values by scaling the texture domain so that its area equals the surface area in 3D. Thus, 1.0 is a lower bound for either norm on any parametrization. Alternately, stretch can be normalized without explicitly scaling the texture domain by multiplying with the factor

$$\sqrt{\sum_{T_i \in M} A(T_i) \Big/ \sum_{T_i \in M} A'(T_i)} \ .$$

To minimize our nonlinear metrics $L^2(M)$ and $L^\infty(M)$, we begin with a uniform-edge-spring solution, and then perform several optimization iterations. Within each iteration, we consider vertices in decreasing order of neighborhood stretch. For each vertex, we perform a line search minimization along a randomly chosen search direction in the (s,t) parametric domain. Because all other vertices are held fixed, the stretch metric only needs local computation over the neighborhood. Since the metric is infinite for degenerate or flipped triangles, the line search is naturally constrained within the kernel of the vertex's neighborhood. In successive iterations, we gradually decrease the convergence tolerance in the 1D line search using the schedule $1/i$ where i is the iteration number. For the L^∞ optimization, we obtain even better results by using our L^2 solution as its starting point.

Figure 1 compares our metric with alternatives. For each metric, we used the same optimization procedure described above. In all cases, boundary vertices were fixed by arc-length. For each parametrization, we created a 128×128 image in the texture domain by sampling a procedural 3D checkered pattern on the parametrized surface. For improved filtering, we used 4x4 supersampling. As can be seen in the resulting textured models, parametrizations optimized using our metrics are better at capturing high-frequency detail everywhere over the surface. In (b-d), note the loss of resolution on the ears where stretch error is high.[2] The area-preserving parametrization in (e) minimizes the Maillot buckling term only. Although it has better spatial distribution than (b-d), note how it undersamples certain directions, causing directional blur on the chin, sides, and ears.



(a) exact

(b) Floater [5] L^2=2.26 L^∞=9.86

(c) MIPS [11] L^2=2.31 L^∞=7.46

(d) Maillot [20] L^2=∞ L^∞=∞

(e) area-preserving L^2=1.57 L^∞=4.19

(f) our L^2 stretch L^2=1.22 L^∞=2.13

(g) our L^∞ stretch L^2=1.28 L^∞=1.65

Figure 1: Chart parametrization comparison.[1]

[1] The harmonic map [4], not shown, is qualitatively similar to Floater [5] and has slightly worse stretch, L^2=2.28, L^∞=10.07.

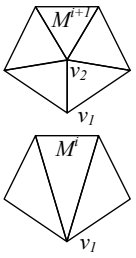
[2] For the Maillot result (Figure 1d), we set the factor α weighting between edge-springs and area-preservation to 0.5. Since the relative scale of 2D to 3D edge lengths is important in this metric, we uniformly scale the 2D domain to have the same area as the 3D chart. Our reported stretch norms are infinite because the minimum solution exhibits buckling. Since our optimization method prevents face flipping, the result is parametrically degenerate triangles having infinite stretch.

4. Our PM parametrization scheme

In this section we present the steps of our PM parametrization scheme. We first introduce some definitions and assumptions.

Let a *chart corner* be any vertex adjacent to 3 or more charts, and let a *chart boundary* be the path of edges separating charts between two corners. We define the *neighborhood* of a vertex as the ring of faces adjacent to the vertex.

Our PM is based on the half-edge collapse operation $(v_1, v_2) \rightarrow v_1$ which affects the neighborhood of v_2 as shown on the right and leaves the position and attributes of v_1 unchanged [16]. We prefer the half-edge to the full-edge collapse to avoid writes to the vertex buffer during runtime LOD changes. Therefore, (s,t) texture coordinates at any vertex must be the same at all LOD levels. Since a vertex on a chart boundary has different (s,t) coordinates on each chart, these must be stored at the corners of mesh faces [10].



To create a texture atlas over a PM, we must enforce the following constraints:

(1) mesh faces cannot span more than one chart, since it is impractical to specify and render disjoint pieces of texture over any single triangle.

(2) chart boundaries must be straight in the parametric domain, since each chart boundary is generally simplified to a single edge in the PM base mesh.

These constraints restrict the partition of the mesh into charts (Section 4.1) and the mesh simplification sequence (Section 4.4).

4.1 Partition mesh into charts

The first step is to partition the mesh into a set of charts: regions with disk-like topology. Ideally one could simultaneously search over the discrete space of possible chart decompositions and the continuous space of parametrizations allowed by each decomposition. Clearly this is intractable. In our system we first partition the mesh using a greedy chart-merging approach. It is similar to simplification schemes based on the greedy growth of "superfaces" [9][15], and to the independent work by Garland et al. [6].

Initially, each face is assigned to be its own chart. For each pair of adjacent charts, we consider the operation of merging the two charts into one, and enter this candidate operation into a priority queue according to a computed cost. As in [6], the merge operation is assigned a cost that measures both its planarity and compactness. We measure planarity as the mean-squared distance of the chart to the best-fitting plane through the chart, defined as a continuous surface integral (unlike [6] which evaluates it only at the vertices). We measure compactness simply as the squared perimeter length.

We iteratively apply the merge operation with lowest cost, and update costs of neighboring candidate merge operations. The process ends when the cost exceeds a user-specified threshold.

A chart merge operation is disallowed if it results in any chart with fewer than 3 corners. It is also disallowed if the boundary between the new chart and any adjacent chart consists of more than one connected component (e.g. one isolated vertex and one path of edges). This constraint also guarantees that charts remain homeomorphic to discs.

Once the charts are formed, they define the set of chart corner vertices. Note that these corner vertices in M^n must appear as vertices in the base mesh M^0 due to the constrained half-edge collapses. Therefore, we would like each chart boundary to closely align with the straight line segment between its adjacent two corners, so as to not be a limiting factor in the simplification quality. We straighten each boundary by computing a shortest path over mesh edges, constrained not to intersect other chart boundaries.

Results of the initial chart partition, and the subsequent boundary straightening are shown in Figure 2. Note that chart boundaries align with important features in the mesh.

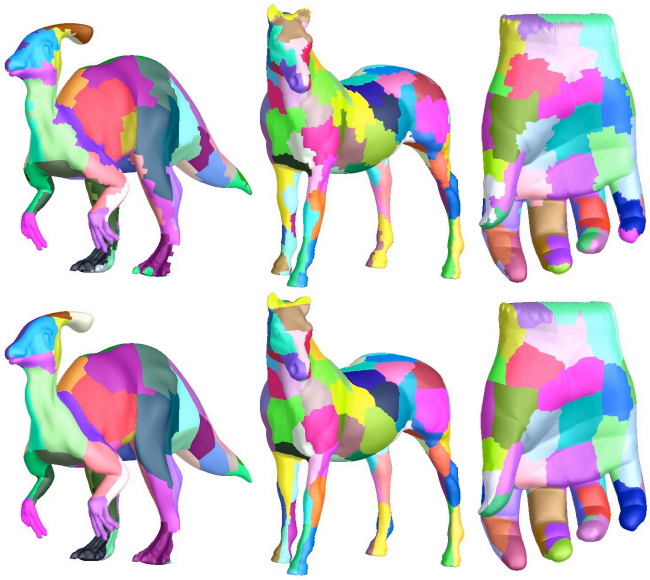


Figure 2: Top row shows initial chart partitions, and bottom row shows result of chart boundary optimization.

4.2 Form initial chart parametrizations

Once the chart boundaries are defined in M^n, we create an initial parametrization of each chart onto a 2D polygon. We define the 2D polygon boundary to be a convex polygon with vertices on a circle, where the length of each polygon edge is proportional to the arc-length of the corresponding chart boundary in 3D, as in [4]. We initially scale the polygon to have unit area. Within each chart, we parametrize the interior vertices by minimizing the $L^2(M)$ stretch metric, using the algorithm described in Section 3.

4.3 Resize chart polygons

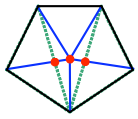
Now that we have chart parametrizations on M^n, we determine how much relative space each chart should be granted in the texture domain. For each chart, we compute $L^2(M^n_{chart})$, the rms stretch over the chart, and use that value to uniformly resize the chart while preserving its shape. We had hoped to use $L^\infty(M^n_{chart})$ to resize the charts, but unfortunately there are a few triangles for which the maximum stretch remains high. Although the relative chart sizes have no effect on simplification (Section 4.4), they do affect $E(PM)$ in the final PM optimization (Section 4.5).

4.4 Simplify mesh

Given the initial chart parametrizations, we simplify the mesh to define a PM. Our goal during simplification is to minimize texture deviation. Our algorithm closely follows that of Cohen et al. [3]. We select edge collapses that minimize texture deviation by using a priority queue. To enforce chart compliance, we disallow an edge collapse $(v_1,v_2) \rightarrow v_1$ in $M^{i+1} \rightarrow M^i$ if vertex v_2

is a chart corner (to preserve corners), or if v_2 is on a chart boundary and edge (v_1,v_2) is not on a chart boundary (to preserve boundary straightness). In addition, we also prevent the creation of parametrically flipped or degenerate triangles.

To measure texture deviation for each candidate edge collapse, rather than using conservative bounds as in [3], we use the fast heuristic of measuring the incremental texture deviation $d(M^{i+1},M^i)$ between the two meshes. (The heuristic is akin to the "memoryless" error that has proven effective for geometric simplification [20].) The maximum deviation between M^{i+1} and M^i is known to lie either at the removed vertex v_2 or at an edge-edge intersection point in the parametric neighborhood (e.g. the red points shown in the figure to the right). We confirmed empirically that the incremental deviation heuristic works well by comparing to a slow simplification that orders edge collapses using the true deviation error (between M^i and M^n).



4.5 Optimize chart parametrizations

Having determined the PM simplification sequence, we now re-optimize the chart parametrizations to minimize stretch and deviation on the entire sequence $M^0 \ldots M^n$. Our nonlinear optimization algorithm follows the strategy of moving vertices of M^n one-by-one in the parametric domain as in Sections 3 and 4.2, but using a different objective function.

Our objective function is a weighted sum of the texture stretch and deviation on all meshes $M^0 \ldots M^n$:

$$E(PM) = \sum_{i=0..n} \psi(i) \left[\lambda L^2(M^i)^2 + (1 - \lambda)\, d(M^i,M^n)^2 / A'(M^n) \right]$$

where $L^2(M^i)$ is the normalized average stretch of M^i (Section 3) computed using the resized charts from Section 4.3, $d(M^i,M^n)$ is its texture deviation, the parameter $0 \le \lambda \le 1$ is used to weight stretch error relative to deviation error, and $\psi(i)$ is the relative weight assigned to each LOD mesh in the sequence. Dividing by the mesh surface area $A'(M^n)$ makes the second term scale-invariant like the first term.

We now introduce a model for setting the relative weight $\psi(i)$ assigned to each mesh M^i, consisting of two factors: usage and scale. Depending on the application, other weighting schemes could be used, without changing the optimization method.

- In LOD applications, coarser meshes are likely to be used proportionately more often. For example, meshes with 10–100 faces are likely to be used more than those with 900–990 faces. We believe that a reasonable model for **usage** probability is a uniform distribution over a *logarithmic* scale of model complexity, e.g. meshes with 10-100 faces are as equally likely as meshes with 100-1000 faces. This distribution is obtained using the factor $1/|M^i|$ where $|M|$ is the number of vertices in M.
- The fact that coarser meshes are typically used when the object is farther away reduces the screen-space **scale** of their deviation and stretch. For a smooth spherical surface, texture deviation varies as $1 / |M|^2$. Since LOD algorithms attempt to maintain a constant screen-space error, deviation and stretch in model space should therefore be down-weighted for coarser mesh meshes using the weighting factor $|M^i|^2$ in $\psi(i)$.

To optimize the texture coordinates of a given vertex v, our optimization algorithm needs to repeatedly evaluate E. Computing E using the above sum over all meshes would be expensive. Fortunately, the neighborhood of v changes only a few times within the sequence of meshes, generally $O(\log |M^n|)$ times. Thus we only need to consider E on each refinement neighborhood $M^i \rightarrow M^{i+1}$ of which v is a member. For each vertex v, we gather the relevant refinement neighborhoods as a list during a coarse-to-fine preprocess traversal of the PM.

Since the refinement neighborhoods adjacent to a vertex v have an approximately logarithmic distribution over the PM sequence, we can account for the usage factor by summing the stretch and deviation on these refinement neighborhoods. Therefore, we weight the error over each such neighborhood by $\psi'(i) = |M^i|^2$ to account for the remaining scale factor.

The following pseudo-code gives an overview of our algorithm.

```
// Optimize parametrization over whole PM sequence.
procedure parametrize_pm()
   gather_refinement_neighborhoods()    // coarse-to-fine traversal
   repeat
      v = some_vertex_in_mesh(M^n)
      optimize_vertex(v)
   until convergence

// Optimize the parametrization param(v) of vertex v
procedure optimize_vertex(vertex v)
   repeat
      vector dir = random_search_direction()    // in 2D domain
      // perform line search minimization
      repeat
         select float t          // e.g. using binary line search
         param(v) = param(v) + dir * t   // perturb parametrization of v
      until error_over_PM(v) is minimized
   until convergence

// Sum of errors affected by param(v) in all meshes M^0…M^n.
function error_over_PM(vertex v)
   error = 0
   for (vertex w in refinement_neighborhoods(v))
      error += error_over_neighborhood(w, v)
   return error

// Error due to v in neighborhood of w (where w is first introduced)
function error_over_neighborhood(vertex w, vertex v)
   return ψ'(level(w)) *
      [    λ * stretch_error(w, original_neighbors(w), v) +
       (1-λ) * deviation_error(w, original_neighbors(w), v) / A'(M^n)  ]
```

As in Section 4.4, we approximate the deviation error $d(M^i,M^n)$ with the incremental deviation error $d(M^i,M^{i+1})$. Because we define our stretch metric to be infinite when a face is flipped in the parameter domain, stretch minimization prevents parametric flipping in all meshes. One final detail is that we also optimize the parametrization of vertices along chart boundaries. Since these vertices have texture coordinates in two adjacent charts, we must consider the refinement neighborhoods in both charts simultaneously. Specifically, we must constrain the parametrizations to remain on the boundaries, and optimize over shared barycentric coordinates along the boundary to prevent "parametric cracks".

4.6 Pack chart polygons

Since the optimization in Section 4.5 modifies the parametrization, we perform a final chart resizing step as in Section 4.3.

The next step is to pack these resized charts into a rectangular texture image. In the context of texture mapping, various heuristics have been presented for the special case of packing 3-sided charts [2][22][25][27]. However, our chart boundaries can be arbitrary polygons. The general problem is known as the NP-hard *pants packing* problem [23].

We simplify the problem by conservatively approximating each chart polygon with the least-area rectangle that encloses it. This rectangle is found efficiently by considering each edge of the polygon's convex hull. Fortunately, our chart polygons are

reasonably shaped, so the rectangle approximation is not too costly. We rotate the chart to align the long axis of the rectangle with the vertical direction. The problem then becomes that of rectangle packing, which is still NP-hard, but for which there exist good heuristic approximations (e.g. [14][24]). We develop our own simple heuristic, which works as follows.

We sort the rectangles by height. In order of decreasing height, we place the rectangles sequentially into rows in alternating left-to-right and right-to-left order as shown in Figure 4c [14]. Through binary search, we optimize over the texture width such that the packing minimizes the area of the enclosing square.

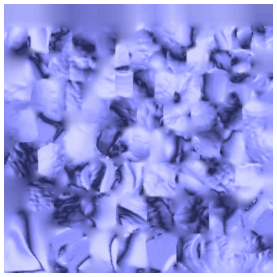
When the desired texture sampling density is later determined, we leave a one texel gap between adjacent charts. Section 5 reports results of our chart packing efficiency.

4.7 Sample texture images

The packed charts define a texture atlas for the surface. We use the atlas to sample attributes from the surface M^n into the texture domain, at the 2D grid of texel locations. For improved filtering, we supersample the attributes using a 4x4 box filter. Section 5 shows results of sampling colors and normals.

If the highest frequency f of the attribute function over the surface mesh is known, the stretch-based scale of the texture atlas makes it possible to estimate the required 2D grid sampling density. With the charts resized as in Section 4.3, the 2D grid spacing should be set no more than $1/(2f)$.

In general, schemes that pack multiple charts into a single texture image may give rise to mip-mapping artifacts, since coarser mip-map levels will average together spatially disjoint charts. The most immediate artifact is that chart boundaries are revealed if the inter-chart area is left unpainted (e.g. black). To mitigate this, we apply a pull-push algorithm [7] to fill-in these unsampled regions with reasonable values. As an example, the effect on the atlas image from Figure 4c is shown on the right.



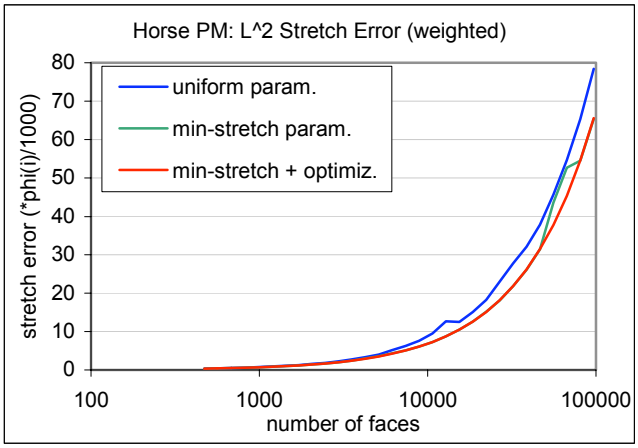
5. Results

Figure 4 shows an example result, where the texture image captures pre-shaded colors from the original mesh M^n. Although we only show the textured base mesh, the same texture atlas can of course be used on all other meshes $M^1 \ldots M^n$ in the PM, as shown on the accompanying video.

Figure 5 shows several mesh approximations in a PM sequence, where the texture image captures a normal map. Because the PM meshes can have irregular connectivities, they quickly converge to good geometric approximations. Hence the figure shows LOD meshes with relatively low face-counts, compared to the original mesh of nearly 97,000 faces.

Figure 3 compares graphs of texture stretch and deviation for meshes in a PM using various parametrization schemes. The curve labeled "uniform" corresponds to uniform edge-spring parametrization followed by simplification minimizing texture deviation. (The harmonic [4] and Floater [5] parametrizations typically have even greater stretch than the uniform parametrization.) The curve labeled "min-stretch param." replaces the initial parametrization with our scheme of Section 4.2. As is evident in the graph, parametric stretch is reduced for the finest mesh M^n. (This difference is often more significant as shown in Table 1.) The curve may appear bumpy because stretch is ignored during simplification. Finally, the curve labeled "min-stretch + optimiz." adds our parametrization optimization of Section 4.5. Note that it improves stretch at lower LODs, while also improving texture deviation over the whole range.

As demonstrated in Figure 6, ignoring texture stretch during parametrization results in non-uniform surface sampling, which becomes apparent as loss of detail over regions of high stretch distortion.



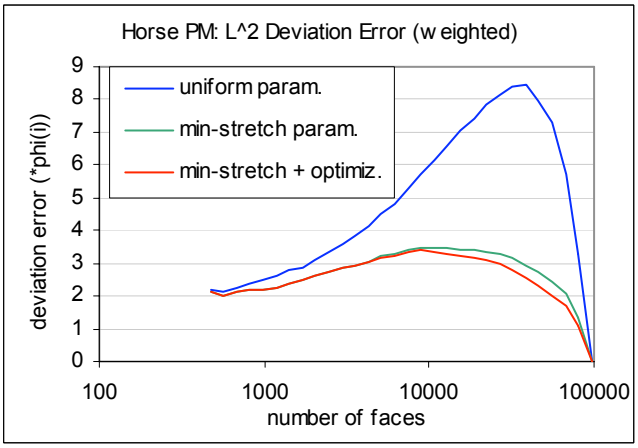


Figure 3: Error graphs of texture stretch and deviation over the horse PM sequence, both measured using L^2 norms and weighted by the $\psi(i)$ factor from Section 4.5.

Models	bunny	parasaur	horse	hand
# faces in M^n	69,630	43,866	96,956	60,856
# vertices in M^n	34,817	21,935	48,480	30,430
# charts	75	75	120	60
# faces in M^0	288	298	470	230
# vertices in M^0	146	151	237	117
(stretch efficiency with uniform parametrization)	0.41	0.001	0.38	0.07
stretch efficiency	0.60	0.40	0.55	0.46
intra-rectangle efficiency rectangle-packing effic. packing efficiency	0.77 0.87 0.67	0.71 0.89 0.63	0.77 0.91 0.70	0.76 0.82 0.62
texture efficiency	0.40	0.25	0.38	0.29

Table 1: Quantitative results.

Table 1 provides results on the efficiency of the parametrization in reducing the required texture memory. *Stretch efficiency* is the total surface area in 3D divided by the total chart area in 2D, $\Sigma_T A'(T) / \Sigma_T A(T)$, given that charts are resized as in Section 4.3. It is less than unity if some surface regions are sampled more than necessary (i.e. if texture stretch is not uniform everywhere and in every direction). *Packing efficiency* is the sum of chart areas in 2D divided by the rectangular texture domain area. It is less than unity due to two factors: the enclosure of chart polygons into rectangles, and the wasted space between the packed rectangles. *Texture efficiency* is the product of stretch and packing efficiencies, or total surface area divided by texture domain area.

A 1-texel gutter is required between texture charts in the texture domain. The overhead of these gutters depends on the resolution assigned to the texture. The packing efficiencies reported in Table 1 ignore this overhead, and therefore assume a reasonably high sampling rate.

Stretch efficiency can of course be improved by partitioning the surface into more charts, but this increases the complexity of the coarsest LOD mesh, and may lower overall texture efficiency due to the additional gutter area. Our stretch-minimizing parametrization allows larger charts with fewer undersampling artifacts.

6. Summary and future work

We have presented a scheme for defining a texture atlas parametrization over the PM representation of an arbitrary mesh. This atlas permits the same texture image(s) to be used for all LOD mesh approximations in the PM sequence. In forming the parametrization, we optimized for both texture stretch and deviation on all meshes in the sequence. We demonstrated that optimizing our new stretch metric creates a balanced parametrization that attempts to prevent undersampling at all locations and along all directions.

There remain a number of areas for future work:

- Examining how best to address the trade-off between texture quality (stretch) and geometric quality (deviation).
- Constraining anisotropy in the parametrization.
- Applying our stretch-based parametrization approach to other multiresolution frameworks such as those using subdivision.
- Speeding up the parametrization optimization of Section 3 using a hierarchical coarse-to-fine approach as in [12].
- Given a known texture, optimizing the parametrization to consider local texture frequency content, as in [13][26].
- Addressing the problems involved when mip-mapping texture images containing multiple charts.
- Considering out-of-core execution for complex models.

Acknowledgments

We would like to thank the MIT Laboratory for Computer Science for use of their equipment. We also thank Stanford University and Viewpoint for the models used in our experiments. The first and third authors were supported in part by grants from the NSF, Sloan Foundation, and Microsoft Research.

References

[1] ABADJEV, V., DEL ROSARIO, M., LEBEDEV, A., MIGDAL, A., AND PASKHAVER, V. Metastream. *VRML 1999 Proceedings*, pp. 53-62.

[2] CIGNONI, P., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. A general method for recovering attribute values on simplified meshes. *IEEE Visualization 1998*, pp. 59-66.

[3] COHEN, J., OLANO, M., AND MANOCHA, D. Appearance-preserving simplification. *SIGGRAPH 1998*, pp. 115-122.

[4] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution analysis of arbitrary meshes. *SIGGRAPH 1995*, pp. 173-182.

[5] FLOATER, M. Parametrization and smooth approximation of surface triangulations. *CAGD* 14(3), pp. 231-250, 1997.

[6] GARLAND, M., WILLMOTT, A., AND HECKBERT, P. Hierarchical face clustering on polygonal surfaces. *Symposium on Interactive 3D Graphics 2001*, pp. 49-58.

[7] GORTLER, S., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. The Lumigraph. *SIGGRAPH 1996*, pp. 43-54.

[8] GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal meshes. *SIGGRAPH 2000*, pp. 95-102.

[9] HINKER, P., AND HANSEN, C. Geometric optimization. *IEEE Visualization 1993*, pp. 189-195.

[10] HOPPE, H. Progressive meshes. *SIGGRAPH 1996*, pp. 99-108.

[11] HORMANN, K., AND GREINER, G. MIPS – an efficient global parametrization method. Technical Report 27/1998, Universität Erlangen-Nürnberg.

[12] HORMANN, K., GREINER, G., AND CAMPAGNA, S. Hierarchical parametrization of triangulated surfaces. *Vision, Modeling, and Visualization 1999*, pp. 219-226.

[13] HUNTER, A., AND COHEN, J. Uniform frequency images: adding geometry to images to produce space-efficient textures. *IEEE Visualization 2000*, pp. 243-250.

[14] IGARASHI, T., AND COSGROVE, D. Adaptive unwrapping for interactive texture painting. *Symposium on Interactive 3D Graphics 2001*, pp. 209-216.

[15] KALVIN, A., AND TAYLOR, R. SuperFaces: Polyhedral approximation with bounded error. *SPIE Proceedings 2164*, pp. 2-13, 1994.

[16] KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. A general framework for mesh decimation. *Proceedings of Graphics Interface '98*, pp. 43-50.

[17] KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. *SIGGRAPH 1996*, pp. 313-324.

[18] LEE, A., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution adaptive parametrization of surfaces. *SIGGRAPH 1998*, pp. 95-104.

[19] LÉVY, B., AND MALLET, J.-L. Non-distorted texture mapping for sheared triangulated meshes. *SIGGRAPH 1998*, pp. 343-352.

[20] LINDSTROM, P., AND TURK, G. Fast and memory efficient polygonal simplification. *IEEE Visualization 1998*, pp. 279-286.

[21] MAILLOT, J., YAHIA, H., AND VERROUST, A. Interactive texture mapping. *SIGGRAPH 1993*, pp. 27-34.

[22] MARUYA, M. Generating texture map from object-surface texture data. *Computer Graphics Forum (Proceedings of Eurographics '95)* 14(3), pp. 397-405.

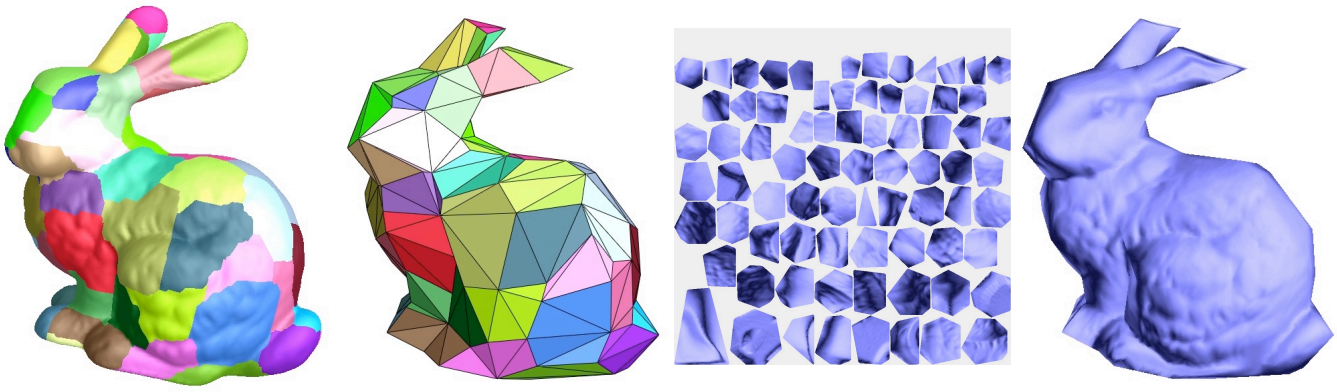
[23] MILENKOVIC, V. Rotational polygon containment and minimum enclosure. *Proc. of 14th Annual Symposium on Computational Geometry*, ACM, 1998.

[24] MURATA, H., FUJIYOSHI, K., NAKATAKE, S., AND KAJITANI, Y. Rectangle-packing-based module placement. *IEEE ICCAD 1995*, pp. 472-479.

[25] SANDER, P., GU, X., GORTLER, S., HOPPE, H., AND SNYDER, J. Silhouette clipping. *SIGGRAPH 2000*, pp. 327-334.

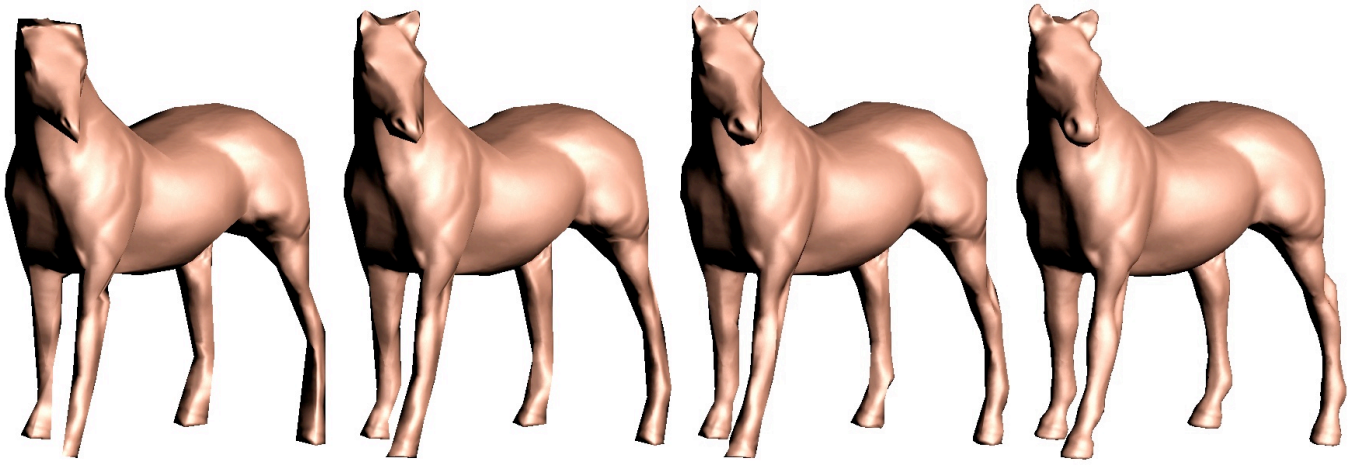
[26] SLOAN, P.-P., WEINSTEIN, D., AND BREDERSON, J. Importance driven texture coordinate optimization. *Computer Graphics Forum (Proceedings of Eurographics '98)* 17(3), pp. 97-104.

[27] SOUCY, M., GODIN, G., AND RIOUX, M. A texture-mapping approach for the compression of colored 3D triangulations. *The Visual Computer* 12, pp. 503-514, 1986.



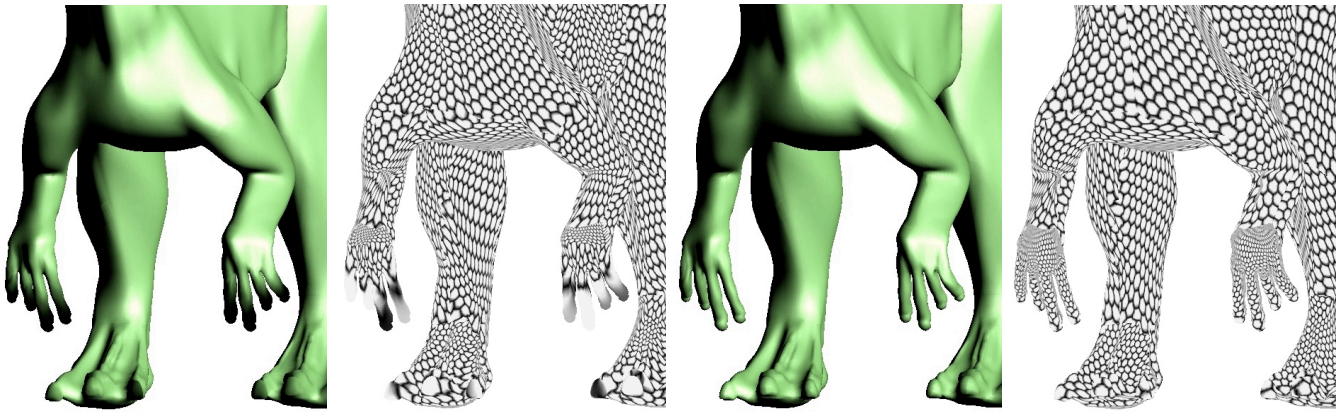
(a) charts on original mesh M^n (b) base mesh M^0 (c) texture atlas (before pull-push) (d) textured base mesh

Figure 4: Overview of our process. We partition the original mesh into charts, establish a stretch-minimizing parametrization on each chart, and simplify the mesh while minimizing texture deviation. With the resulting PM sequence $M^0 \ldots M^n$, we further optimize the parametrization to reduce stretch and deviation in all meshes. Finally, we pack the charts into an atlas, and fill the atlas with texture samples from M^n.



(a) M^0 (470 faces) (b) M^{115} (700 faces) (c) M^{365} (1,200 faces) (d) M^{4765} (10,000 faces)

Figure 5: Textured mesh approximations in a PM sequence. All meshes refer to the same 512x512 texture atlas.



sampled normal map hex grid in 2D texture domain sampled normal map hex grid in 2D texture domain

(a) ignoring stretch (b) using our scheme

Figure 6: Texture stretch illustrated using both a sampled normal map and a uniform honeycomb pattern overlaid in the 2D texture image. (a) Charts are parametrized using uniform edge weights and scaled by surface area. (b) Our scheme considers texture stretch when both parametrizing and scaling the charts. In (a), note the loss of fine detail in regions with stretched honeycomb pattern (e.g. fingers and toes).